並列処理によるリアルタイム・レンダリング・システムの開発事例

Realtime Rendering Application Development using Parallel Processing

西 川　孝, 西　俊 郎

要　約　CADで生成した面形状の品質評価を行う目的で，レイトレーシングによるリアルタイム・レンダリング・システムを開発した．ユーザの操作に，リアルタイムに追従するためには，1秒間に10枚のレンダリング画像を生成し，CAD上で連続的に表示しなければならない．既存のシステムでは，1枚の画像を生成するのに8分程度かかっていたので，ハードウェアの選定やソフトウェアの効率改善で，約4,800倍の高速化を目標として開発を進めた．その結果，1秒間に10枚のレンダリング画像を生成することができた．本稿は，開発にあたり，レンダリングの並列処理，画像データの圧縮・解凍や転送についての改善事例を報告する．

Abstract　To evaluate the surface smoothness and continuity of geometric modeling designed by CAD (computer aided design), the rendered image using ray tracing technique is much recommended. It only generates more realistic image. But one of disadvantage is very cost expensive. The existing system took approximately 8 minutes to create a single rendered image, so that it is hard to use this technique interactively. On the other hand, the clients would want to check the modeling image in a different angle so quickly. To meet their requirements, the rendering performance is requested up to 10 frames per second. It means the target system must tune up more than 4800 times faster. To make this solution, the parallel processing machine with 72 CPUs was introduced. Therefore, to achieve the target, we applied many tuning-up actions such as reducing the traffic rate between the client and parallel processing machine, keeping the balance of calculation of each processing unit and so on. This paper discusses how to improve the rendering performance with keeping the rendering quality.

1. はじめに

近年，CG（コンピュータ・グラフィックス）技術は，映画，コマーシャル・フィルム，アニメーション，ゲーム，アートなど，私たちの身近なところで，様々な用途として利用されている．製造業の分野では，物理シミュレーションを目的として，CG技術が製品開発に密接に関わってきた．例えば，意匠デザインの工程では，従来，実物大のモデルを試作して，形状の評価や審査・検討を行っていたが，最近では，CG技術で可視化した形状を評価することで，モデルの試作を極力減らし，開発期間の短縮やコスト削減を図っている．

ある顧客は，意匠デザインの工程で自社開発したバッチ処理のレンダリング[*1]・システムを利用していた．このシステムは，実測した物体の色・材質を忠実に再現する着色理論を実装しているため，高精度なレンダリング画像を作成できたが，1枚の画像を生成するのに，おおまかな形状でも8分程度の計算時間がかかっていた．顧客は，CAD（設計支援システム，以後クライアントとする）で生成した面形状の品質評価を行うため，クライアントの操作に追従してレンダリング・表示する，リアルタイム・レンダリング・システムの開発を要望した．

著者らは，リアルタイム・レンダリング・システムの開発にあたり，1） レンダリング機能の効率改善，2） 並列処理，3） 画像の圧縮・解凍，4） 画像のネットワーク通信などの工程で効率化を図った．本稿は，レンダリングの並列処理の改善事例を中心に報告する．

2. システムの概要

2.1 システムの目的

本システムは，既存のレンダリング・システムをもとに，以下の目的で開発を行った．

1） レンダリング機能の品質向上と効率改善

2） クライアントの曲面式移行に伴う図形演算処理ルーチンの改良と効率改善
（3 次曲面式→高次曲面式）

3）クライアントの操作にリアルタイムに追従するレンダリング・システム
（クライアント・サーバー形式で高速に通信する仕組みの確立）

2.2 システム構成

計算サーバーには，各社の最新機種を使用してベンチマークを行った結果，最も速い性能を得た並列処理コンピュータを選択した．このシステムは，一つのノード内に 4 CPU が密結合した共有メモリ型システムとなっていて，ノード間は高速なネットワークで疎結合したクラスタ・システムである．ノード数は最終的に 16 ノードとなり，合計 64 CPU のシステムである．計算サーバーとクライアントは，クライアント・サーバ方式をとり，表示パラメータ，画像データをネットワーク通信する（図 1）．

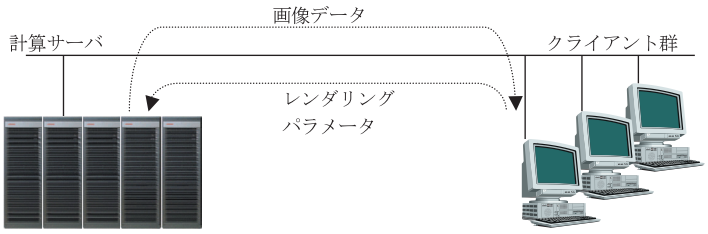

図 1 クライアント・サーバ構成

2.3 システムの目標

本システムの開発には，クライアントの操作に対して「リアルタイムに追従する」という大きな課題があった．リアルタイムに追従するためには，1 秒間に 10 コマ以上のレンダリング画像を，連続的に表示させなければならない．既存のシステムでは，1 コマの作成に 8 分の計算時間がかかっており，新システムは H/W（ハードウェア）の性能向上と，S/W（ソフトウェア）による効率改善を合わせて，約 4,800 倍の高速化が必要となった．

開発当初は，32 CPU の計算サーバでアプリケーションの実験・開発を行い，後に，64 CPU の次期サーバで本番稼働となった．本番機は，CPU 数が 2 倍（64 CPU），1 CPU 当りの性能は 4 倍程度向上する見込みであったため，全体で理論的に 8 倍程度速くなると推定した．

開発と実施にあたり，計算時間（レンダリングの効率改善，並列処理の改善），画像転送（画像圧縮，ネットワークの高速化），画像表示の高速化等など，ある評価モデルを対象とした目

標を，表 1 に示す．

表 1　目標設定

項 目	開発当初(32CPU)	最終目標(64CPU)	改善率
計算時間	6.58 秒	0.10 秒	約 66 倍
画像転送	0.45 秒	0.08 秒	約 5.6 倍
画像表示	0.05 秒	0.03 秒	約 1.7 倍
コマ/秒	0.7 コマ/秒	10 コマ/秒	約 14 倍

2.4　システムの動作手順

2.4.1　レンダリングの開始

クライアントは，リアルタイム・レンダリングのコマンドを起動すると，形状データ（数十 MB）を計算サーバにリモートコピーする．次にソケットでコネクションを確立し，表示パラメータ（回転・移動・拡大，色情報，光源情報など）をデータ送信する．計算サーバは，その形状データと表示パラメータをもとにレンダリングする（図 2）．

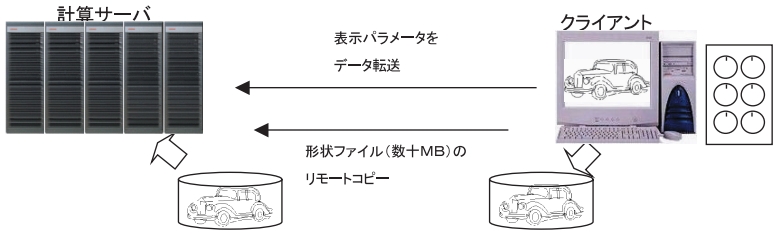

図 2　レンダリング開始

2.4.2　レンダリング画像を表示

計算サーバは，レンダリングした画像を圧縮してクライアントにデータ送信する．クライアントはデータ受信後，圧縮データを解凍しディスプレイに表示する（図 3）．

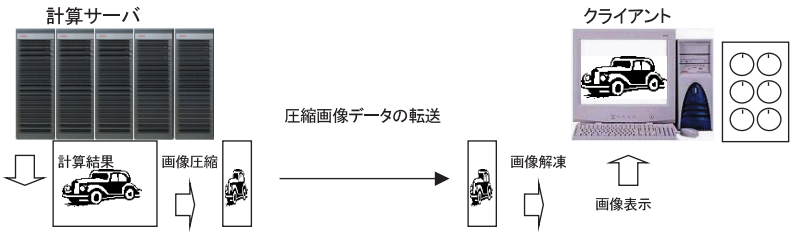

図 3　計算結果を表示

2.4.3　2 回目以降のレンダリング

クライアントは，2 回目以降，形状データの転送を行わずに表示パラメータのみ送信する．例えば，図 4 のように，クライアント上でダイヤルを回しモデルを回転させると，即座に計算サーバに表示パラメータが転送される．計算サーバは，受信した表示パラメータに基づいてレンダリングし，結果画像をクライアントに送信する．

この応答をリアルタイムに実現する（図 4）.

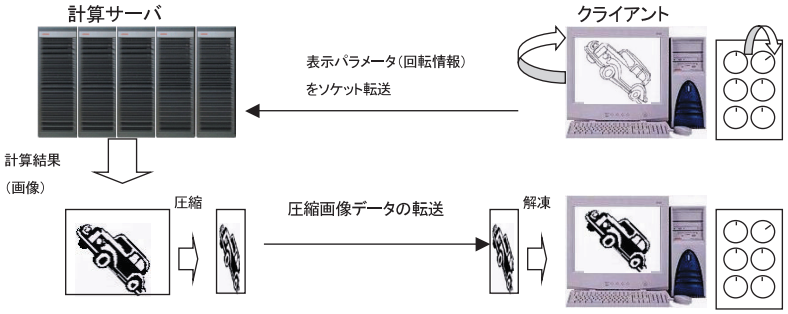

図 4　レンダリング 2 回目以降

3.　並列処理対応

3.1　レイトレーシング

本システムのレンダリング手法は，レイトレーシング法で計算している．レイトレーシングとは，図 5 のように視点からスクリーン上の一つの画素を通って形状モデルに向かう光線を考え，その光線の反射，透過，屈折を追跡する．そしてあらかじめ定義されたモデルの属性とその環境から，注目画素の色を決定する．以降，この作業を全画素に対して行っていくことにより，画像を完成させる．

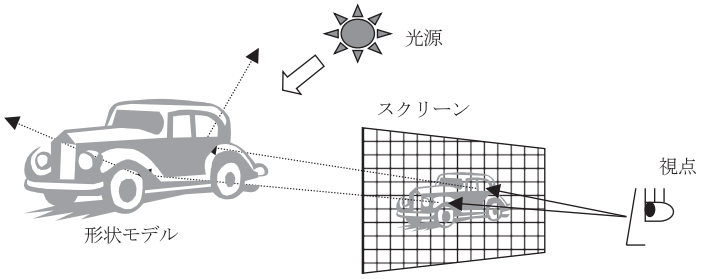

図 5　レイトレーシング

数値計算の観点からみると，レイトレーシングは画素ごとの計算が独立しているという特徴がある．これは，どの画素からどの順序で求めても，等しい結果が得られることを意味する．例えば，図 6 の幅＝MaxX，高さ＝MaxY の画像の作成を擬似コードで表すと，下記のように，一つの画素の色を求める処理を，画像の幅，高さの変数でループさせ，全画素の色を求めることになる．

3.2　並列処理の概要

レイトレーシングによる 1 画素の計算時間を => で表現すると，一つの CPU で逐次的に計算する場合，全画素の計算時間は，図 7（全 8 画素の例）のように => の総和となり，T 2-T 1 時間となる．

レイトレーシングは，画素の計算順序に依存しないので，図 7 の逐次処理を複数の CPU に

```
Y=1 から Y=MaxY まで
    X=1 から X=MaxX まで
        (X,Y)画素の計算
        X = X + 1
    End Loop
    Y = Y + 1
End Loop
```

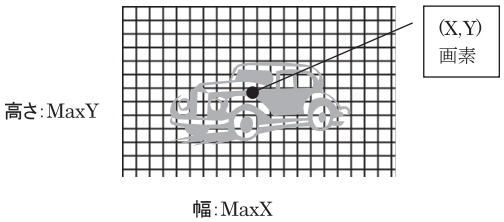

図 6　1 画素の計算

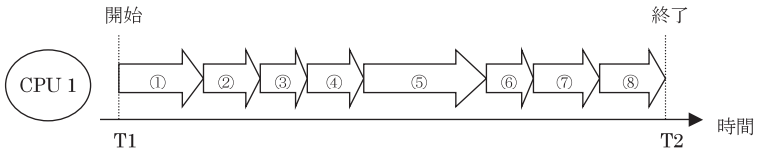

図 7　1 CPU による計算時間（逐次処理）

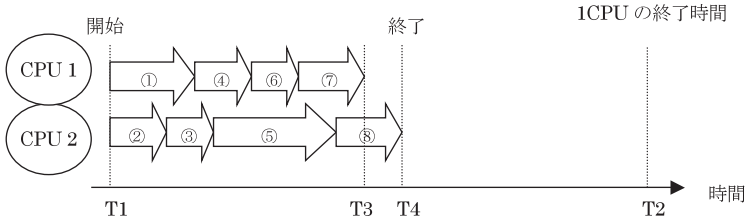

図 8　複数 CPU による計算時間（並列処理）

割り当て，図 8（2 CPU の例）のように，並列に処理することで高速化することができる．

多くの CPU で並列処理すると，計算時間をより短縮できるが，一般的に CPU 数に比例して速くなるとは限らない．理由は，各 CPU での計算量にバラツキがあるためである．図 8 の例では，T 3–T 1 時間は並列処理しているが，T 4–T 3 時間は 1 CPU での計算となり，全体の計算時間は遅い方の T 4–T 1 時間となる．もしも，負荷を各 CPU へ均等に分散できれば，全体の計算時間は，(T 2–T 1)/2＝(T 3–T 1)＋(T 4–T 3)/2 時間と短縮できる．並列処理の効率を上げるためには，各 CPU への負荷分散（ロードバランス）を均一化しなければならない．

3.3　並列処理の実装

並列化の手法は，大きく分けるとマルチスレッド方式，マルチプロセス方式，そして，その組み合せがある．マルチスレッド方式とは，OS（オペレーション・システム）レベルで一つのプロセスを複数のスレッドに分割し，スレッドを CPU に割り当てる方法である．また，マルチプロセス方式は，複数のプロセスを複数の CPU に割り当て，プロセス同士がメッセージ通信しながら，処理を進めていく方法である．どちらの方法が適しているか，また，どの規模まで並列処理を行うかなどは，アプリケーションの性質や H/W 環境によって左右される．

本システムは，移植性，拡張性に有利なマルチプロセスの方法を選択し，メッセージ通信は，業界標準の MPI（Message Passing Interface）を使って並列化した．

マルチプロセスでの並列処理は，一般的なマスタ・スレーブ（Master/Slave）モデルで行

った．マスタ（Master）は，全体の仕事を複数のスレーブ（Slave）に分散処理させ，処理状況を管理しながら全体の仕事を完成させる役割を持つ．スレーブはマスタから指示された範囲の仕事を処理する．具体的には次のように実装した．マスタは，全体の画素をある一定の領域に分割し（例えば，図 9 左図のように 16 分割），スレーブに計算する領域の情報をメッセージ送信する．スレーブはマスタからメッセージ受信した領域の画素を計算し，結果画像をマスタにメッセージ送信する．マスタは，その領域の画像を保管し，次の計算領域をスレーブにメッセージ送信する．これを，全領域の計算が終了するまで行う．

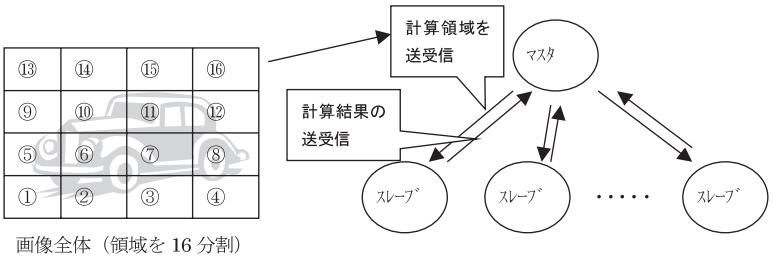

図 9　マスタ（Master）とスレーブ（Slave）の役割

3.4　ロードバランスの最適化

並列処理の性能は主に，下記の二つの要因によって決定される．

1）　ロードバランス（CPU の負荷分散）

2）　メッセージ通信のオーバヘッド

ロードバランスが適切でないと CPU の数を増やしても，CPU がアイドルになる時間が増えるだけで，CPU の数に比例した性能は得られない．また，ロードバランスを良くするために，仕事の単位をより細かくすると，マスタ・スレーブ間のメッセージ通信のオーバヘッドが増し，全体の処理効率を低下させることになる．このバランスは，CPU 間の通信ネットワーク性能やメモリの配置方法などの H/W 環境と，アプリケーションの性質とが密接に関わっている．

本システムの場合，1 枚の画像を高速に作成することを目的として並列処理を行っている．1 枚の画像は，幅　1182 ドット，高さ 804 ドットの約 95 万画素あり，1 画素あたりは R, G, B の 3 Byte 構成なので，データサイズは約 2.9 Mbyte となる．この 95 万回の処理を，複数の CPU にどのように分散すれば最適な効率が得られるかを調査分析した．

マスタ・スレーブ・モデルをノード間でメッセージ通信する条件で，表 2 に示す通信のデー

表 2　メッセージサイズと通信時間の関係

ｽﾚｰﾌﾞの計算画素	通信のデータ量（Byte）	ﾏｽﾀｰ/ｽﾚｰﾌﾞの通信回数	通信に要した時間(sec)	通信ｺｽﾄ(sec)
1	3	950328	3.885742	0.000004
4	12	237582	0.977539	0.000004
16	48	59496	0.250976	0.000004
64	192	14948	0.073242	0.000005
256	768	3774	0.023438	0.000006
1024	*3072*	*962*	*0.010743*	*0.000011*
4096	*12288*	*247*	*0.007812*	*0.000032*
16384	49152	70	0.067382	0.000963

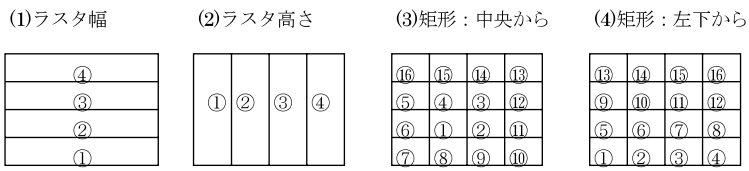

図 10　画像の領域分割方法

タ量と通信回数の組み合せによる全画素のデータが収集する時間を調べると，メッセージ通信のサイズは 3 KByte〜12 KByte の間で，最も効率が良いことが判った．このサイズは，画素数にすると，1024〜4096 画素に相当する．

次に，図 10 のように，スレーブに分散する領域の分割方法，順序を変えて，よりロードバランスの良い分散方法を調査した．

その結果，表 3 に示すように，矩形（32×32）で分割した領域を，中央部分から順にスレー

表 3　各種分散方法による計算時間の比較

（1）　ラスタ：幅

幅	高さ	ｽﾚｰﾌﾞの計算画素	通信のデータ量(Byte)	ﾏｽﾀﾞ/ｽﾚｰﾌﾞの通信回数	計算時間
1182	1	1182	3546	804	2.924
1182	2	2364	7092	402	1.755
1182	4	4728	14184	201	0.950
1182	8	9456	28368	100.5	0.667

（2）　ラスタ：高さ

幅	高さ	ｽﾚｰﾌﾞの計算画素	通信のデータ量(Byte)	ﾏｽﾀ/ｽﾚｰﾌﾞの通信回数	通信+計算時間
1	804	804	2412	1182	2.947
2	804	1608	4824	591	1.814
4	804	3216	9648	295.5	0.992
8	804	6432	19296	147.75	0.682
16	804	12864	38592	73.875	0.630

（3）矩形：中央から

幅	高さ	ｽﾚｰﾌﾞの計算画素	通信のデータ量(Byte)	ﾏｽﾀ/ｽﾚｰﾌﾞの通信回数	通信+計算時間
1	1	1	3	950328	9.653
2	2	4	12	237582	4.398
4	4	16	48	59496	1.688
8	8	64	192	14948	0.838
16	16	256	768	3774	0.527
32	*32*	*1024*	*3072*	*962*	*0.406*
64	64	4096	12288	247	0.439
128	128	16384	49152	70	0.444

（4）矩形：左下から

幅	高さ	ｽﾚｰﾌﾞの計算画素	通信のデータ量(Byte)	ﾏｽﾀ/ｽﾚｰﾌﾞの通信回数	通信+計算時間
1	1	1	3	950328	9.912
2	2	4	12	237582	4.438
4	4	16	48	59496	1.697
8	8	64	192	14948	0.845
16	16	256	768	3774	0.536
32	32	1024	3072	962	0.442
64	64	4096	12288	247	0.444
128	128	16384	49152	70	0.572

ブに分散する方法が，ロードバランスが最も効率が良いことが判明した．（本システムで採用）

一般的に，分散する仕事の計算量の差が大きい場合，図 11 のように，計算量の多い部分から順にスレーブへ分散した方が，（T 3–T 2）の時間が短くなり，全体の効率は良くなる．

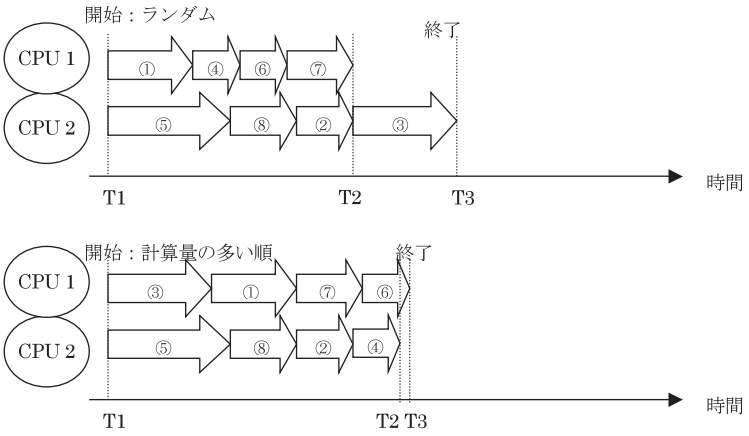

図 11　分散順序の違いによる処理時間の比較

3.5　並列処理の改善

3.5.1　クライアント・サーバの実装

設計当初は，クライアントからの表示パラメータの受信，画像表示プロセスへの画像データ送信は，マスタが行うようにしていた（図 12）．

レイトレースの処理時間は，効率改善により 130 msec にまで短縮されたが，この実装方法では，クライアント側で体感する待ち時間は，195 msec（5 コマ/秒）と満足する追従性は得

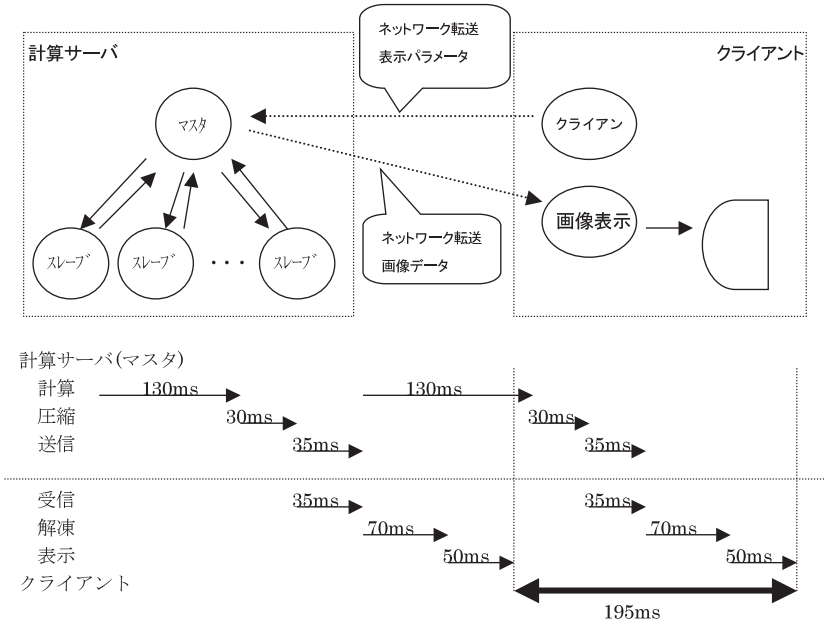

図 12　クライアント・サーバの同期通信

られなかった．目標を達成すべく実験解析を行い，以下の対策を実施することにより成果を挙げた．

3.5.2　クライアントとの非同期通信

上記の実装方法の問題点を解析した．レイトレースの処理と画像圧縮・データ送信の処理は，マスタが行っている．したがって，マスタが画像転送している間は，スレーブに次の仕事の指示が出来ず，スレーブ側はアイドル状態が発生する．その状態を回避するため，新たに画像圧縮とデータ送信を専用に行う，「転送」プロセスを生成することにした．マスタに集まる画像データは，「転送」プロセスからも参照できるように，同じノードの共有メモリに格納し，データを転送している間，マスタは次々と先の計算が出来るようにした．この改善により，マスタは画像の転送を待たずに，クライアントからの計算要求を連続的に処理できるようになり，195 msec（5 コマ/秒）が 155 msec（6.5 コマ/秒）に改善された（図 13）．

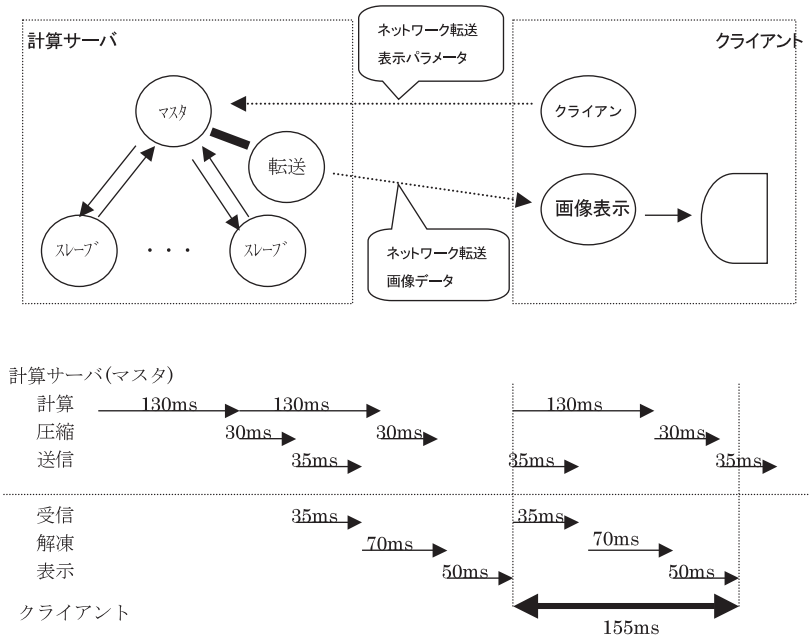

図 13　クライアント・サーバの非同期通信

3.5.3　通信オーバヘッド対策

CPU 数が増えると，マスタ・スレーブ間の通信で衝突が発生する確率が高くなり，並列処理の効率は徐々に低下する．この現象を緩和するために，今までマスタに画像を集めていたのを止め，画像データを収集する専用のプロセスを生成した．その結果，メッセージは，マスタ→スレーブ→「画像収集」プロセスという流れになり，マスタでの通信の一局集中を回避することができた．

計算サーバ内のテストでは，レイトレースの計算時間が，130 msec から 100 msec へと改善された．また，クライアント側で，受信と表示が並列に処理できるように，2 CPU のマシンで運用した結果，全体で 105 msec（9 コマ/秒）となり，追従性が大きく向上した（図 14）．

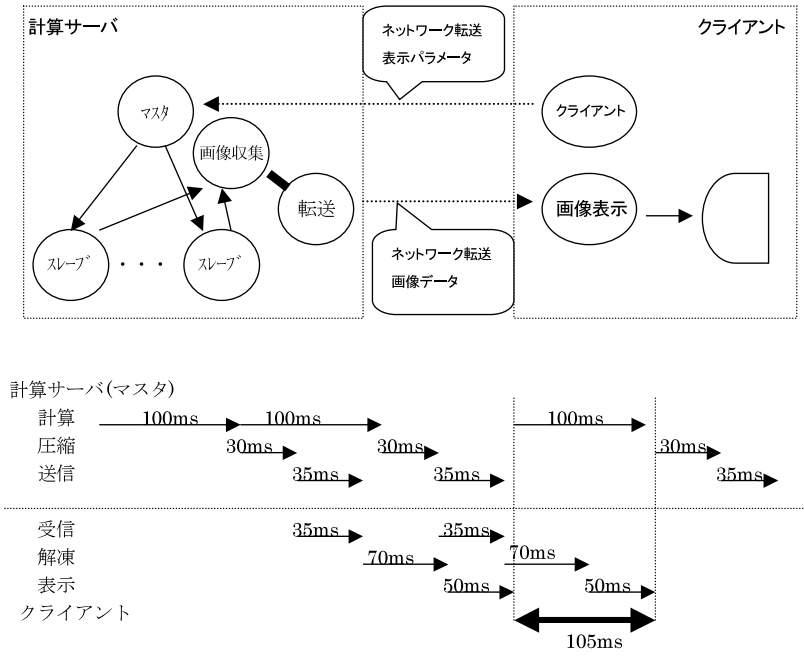

図 14　マスタ・スレーブ・モデルの改善

3.6 障　害　対　策

一般的に，コンピュータはノード数，CPU 数の増加に伴い，故障率が高くなる傾向にある．本システムの並列処理コンピュータは，16 ノード，64 CPU で構成されており，実際，長時間連続実行中に，ある CPU に異常が発生すると，その CPU が含まれるノードが H/W 障害でダウンする場合がある．

本システムは，高精度な静止画や動画を作成する場合，バッチ処理で行う．利用者は，計算時間のかかるレンダリング処理を，休日にかけて一括にバッチ投入し，休日空けにその結果を利用する場合が多い．よって休日中にシステムがダウンし，アプリケーションが停止することは，作業スケジュールが遅れるほど被害が大きい．それを回避するために次の対策を実施した．

3.6.1　障害発生時の処理の継続実行

レンダリング実行中に，あるノードがダウンすると，ダウンしたノードのプロセスから応答が得られないため，その先の処理が止まっていた．この状態を回避するために，マスタとスレーブの通信を全て非同期通信に変更した．マスタは，スレーブからある一定時間内にメッセージの応答がない場合，そのプロセスがダウンしたとみなし，そのスレーブに与えた仕事を他のスレーブに計算させた．この改善により，一部のノードのダウンにも支障なく，レンダリングを完成することができた．

3.6.2　障害時のジョブの連続実行

障害が発生した後，キューイングされているジョブを実行する前に，応答のある CPU を調べ，使用可能な CPU のみジョブを起動するように改善した．これにより，あるノードがダウンした後でも，適切な CPU 数でキューされているジョブの連続実行が可能となった．

3.6.3 チェックポイント・リスタート機能

今までは，レンダリング処理を中断した後，そのジョブをやり直すと，最初から計算をやり直さなければならなかった．途中まで計算した結果を再利用するために，チェックポイント・リスタート機能を付加した．この機能は，レンダリング計算情報を一定の時間間隔で出力し，ジョブを中断したときや，サーバに障害が発生したときに，チェックポイントまでの計算結果を再利用し，残りの部分のみを計算することで，再開時のスループットの向上を図った．

4. 画像データ転送対応

4.1 画像圧縮・解凍，画像表示

計算サーバで生成される画像データは，約 2.9 MB あり，LAN 経由でクライアントへ転送される．一般的な 100 Base Ethernet の LAN で，2.9 MB のデータを計算サーバからクライアントに転送するには，260 ms かかる．

データの転送時間の短縮を図るため，圧縮率だけに着目するのではなく，トータルの画像圧縮＋ネットワーク転送＋画像解凍＋画像作成＋画像表示の合計時間が最も短く，かつ，元の画像の品質を落とさないという条件で，各圧縮方法を評価した（表 4）．

表 4　圧縮方法の違いによる各種性能比較（100 Base Ethernet 使用）

（単位：msec）

圧縮方法		圧縮率	圧縮	転送	解凍	作成	表示	合計
圧縮なし	OpenGL			257		93	76	426
	XLib			257		68	148	473
JPEG 圧縮	OpenGL	4.0 %	798	13	457	95	75	640
	XLib		798	13	457	68	146	684
LZO 圧縮	OpenGL	9.6 %	363	27	196	95	75	393
	XLib		363	27	196	69	148	439
ランレングス点順次	OpenGL	17 %	73	46	87	94	76	302
	XLib		73	46	87	68	147	348
ランレングス面順次	OpenGL	9.5 %	160	27	145	95	75	342
	XLib		160	27	145	68	148	388

上記の結果より，100 Base Ethernet では，画像をランレングス点順次圧縮して，ネットワーク転送するのが，最も速かった．よって，ランレングス点順次圧縮を採用した．

ランレングス点順次圧縮のアルゴリズムは，画素（R, G, B）の値を，次の画素と比較し，R, G, B の値が全て等しい場合，その画素と同一のものとして個数情報を付加する．（L が個数）

（R 1, G 1, B 1),（R 2, G 2, B 2),.....・(Rn, Gn, Bn)画像データ列は，（R 1, G 1, B 1, L 1),.....,(Rm, Gm, Bm, Lm）となる．この圧縮方法の特徴は，同じ色（R, G, B）が連続するほど圧縮率が良くなるが，同じ色が連続しない場合，むしろ，レングス情報の分だけ，元のデータより増えることになる．

本システムでは，背景に画像を合成する場合があり，このときには，ランレングス圧縮を使うと，逆に元のデータよりデータ量が増えることがある．したがって，背景合成の有無によって，圧縮する・しないの処理を行った．

4.2 ジャンボ・パケット転送

一般的な TCP/IP 通信では，MTU(Maximum Transfer Unit)のデフォルト値が，1,500 Byte

に設定されている．MTU とは，通信インタフェースが一度に転送できる最大の量で，この値を超えたデータを転送する場合，分割して転送される．計算サーバとクライアントとは，3 MB 弱の画像データを通信しているので，MTU を設定可能な最大値の 9,000 Byte に設定した．この変更で，データ転送時間を約 70% 効率改善できた．

5. おわりに

5.1 評　価

評価モデルを対象とした，効率改善の成果を表 5 に示す．計算時間に関しては，レイトレースの効率改善，並列処理の改善，計算サーバーの強化で，0.10 秒を達成できた．

表 5　改善成果

項 目	目標値	改善前	改善後	達成度
計算時間	0.10 秒	6.58 秒	0.10 秒	◎
画像転送	0.08 秒	0.45 秒	0.11 秒	△
画像表示	0.03 秒	0.05 秒	0.05 秒	△
コマ/秒	10 コマ/秒	0.7 コマ/秒	9 コマ/秒	○

本システムのような，大規模に並列処理コンピュータを使って，CAD の操作に追従するリアルタイム・レンダリング・システムは稀である．本システムの開発には，何回も実験や試行錯誤を繰り返しながら，いかに並列処理の計算能力を発揮させるかの工夫を進めた．今回の報告が，並列処理向けシステム開発における，何かのヒントになれば幸いである．

5.2 今後の課題

開発当初は，少数の面を対象として，リアルタイム性を追求してきたが，最近では，面数や部品数などのデータ量が増え，さらに，映り込み，背景合成，ガラス，メッキ塗装，厳密な影表現等など，より現実に近い環境下で形状モデルの評価が求められている．今後，データ量や計算量は，確実に増える傾向にある．よって，大量データの対応や，社内ネットワークに接続する PC を一つの計算ノードとして，粗密混在型のグローバルな計算資源で並列処理するシステムを考えていきたい．

また，最近，リアルタイム・レンダリングのために，プログラム可能なグラフィックス・ハードウェアが登場してきた．レイトレーシング並みのレンダリング品質は表現できないものの，以前のグラフィックス・ハードウェアに比べて，かなり品質の高いレンダリング表現が可能になってきている．今後，この分野にも注目していきたい．

＊ 1　3 次元の形状モデルに対して，光源や周囲の光，視点からの見え具合，物体表面の色や反射率などを考慮して，物体の色・陰影・光沢などを決める手法のこと．

参考文献　［1］ Computer Graphics 技術編 CG 標準テキストブック，CG-ARTS 協会
［2］ 色彩科学ハンドブック，東京大学出版会
［3］ 高橋義造，並列処理機構，電子・情報・通信編，丸善株式会社
［4］ ASC-9310330, the National Science Foundation Science and Technology Center, MPI : Message Passing Interface Forum

［5］ Foley, VanDam, Feiner, Hughes, Computer Graphics

執筆者紹介 西　川　　孝（Takashi Nishikawa）
1971 年東京都立航空工業高等専門学校機械工学科卒業．同年日本ユニシス(株)入社．UNICAD/SOLID，グラフィック・アクセラレータの開発を経て，現在，自動車産業事業部トヨタ統括部 CG システム開発室に所属．

西　　俊　郎（Toshiro Nishi）
1990 年東京学芸大学教育学部数学科卒業．同年日本ユニシス(株)入社．同年より，CG システム等の開発を担当，現在，自動車産業事業部トヨタ統括部 CG システム開発室に所属．